UNIVERSIDAD POLITÉCNICA DE CATALUÑA ESEIAAT

INSTALACIÓN AUDIOVISUAL INTERACTIVA

Convocatoria Mayo 2019

Titulación:

Grado en ingeniería Electrónica Industrial y Automática

Autor:

Oriol Elié Joan

Director:

Jorge Martín Giménez

Codirector:

Rafael Weyler Perez

Fecha:

09/05/2019

Índice

Índice de imágenes

Índice de tablas

Resumen

En este proyecto se presenta el desarrollo i la implementación de un sistema audiovisual interactivo mediante el uso de tecnologías de bajo coste, como puede ser Arduino, Processing y los distintos componentes electrónicos usados.
El trabajo incluye la elección de la tecnología más apropiada, así como la elección de los componentes y su montaje para poder realizar el trabajo de manera satisfactoria.
En la primera parte del proyecto se describe el microcontrolador Arduino usado para enviar, los valores captados en el velocímetro que he creado en Arduino, hacia Processing y que estos valores tengan un efecto en el recorrido en bicicleta creado en Processing. A continuación se explicara y justificara todos los componentes usados para crear el velocímetro de Arduino así como la transmisión de los datos hacia Processing.

Abstract

This project presents the development and implementation of an interactive audiovisual system through the use of low-cost technologies, such as Arduino, Processing and the different electronic components used.

The work includes the choice of the most appropriate technology, as well as the choice of components and their assembly in order to perform the work satisfactorily.
In the first part of the project we describe the Arduino microcontroller used to send, the values captured in the speedometer that I created in Arduino, to Processing and that these values have an effect on the bicycle path created in Processing. The following will explain and justify all the components used to create the Arduino speedometer as well as the transmission of the data to Processing.

Agradecimientos

A Jorge Martin, director de este proyecto, por su ayuda y soporte en la realización de este proyecto.
A mi familia por darme soporte y darme ánimos en el transcurso de este proyecto, y estar a mi lado cuando los he necesitado.
A mis compañeros, José Feria, Raúl Linares, Arnau Gómez, quien me han acompañado a lo largo de todos estos años en la universidad y con quien he pasado grandes momentos, y me han ayudado en los momentos difíciles.

CAPÍTULO 1: INTRODUCCIÓN

1.1 Descripción

La idea de este proyecto ha sido la de crear un programa que nos permita realizar una simulación de un recorrido ya sea por ciudad, en un museo etc., mediante el uso de tecnologías de bajo coste, como pueden ser Arduino, Processing y los distintos componentes electrónicos usados.
En este proyecto se ha decantado por realizar un recorrido por ciudad en bicicleta.
El trabajo incluye la elección de la tecnología más apropiada, los componentes a utilizar y su montaje para poder realizar el trabajo de manera satisfactoria.
En la primera parte del proyecto se describe el microcontrolador Arduino usado para enviar, los valores captados en el velocímetro que he creado, hacia Processing y que estos valores tengan un efecto en el recorrido en bicicleta. A continuación se explicara y justificara todos los componentes usados para crear el velocímetro así como la transmisión de los datos hacia Processing.
En la segunda parte se describirá la parte de programación de Processing, que es la que se encarga de ejecutar el recorrido grabado en videos anteriormente, así como de obtener los datos enviados, por puerto serie, de Arduino que afectarán al recorrido simulado.

1.2 Objetivo

Este proyecto tiene como objetivo desarrollar un control sobre un recorrido virtual en bicicleta por ciudad, pudiendo controlar la velocidad y dirección de este recorrido. Para este proyecto se ha escogido la ciudad de Badalona. En esta imagen podemos observar el recorrido en bicicleta, que se llevara a cabo.

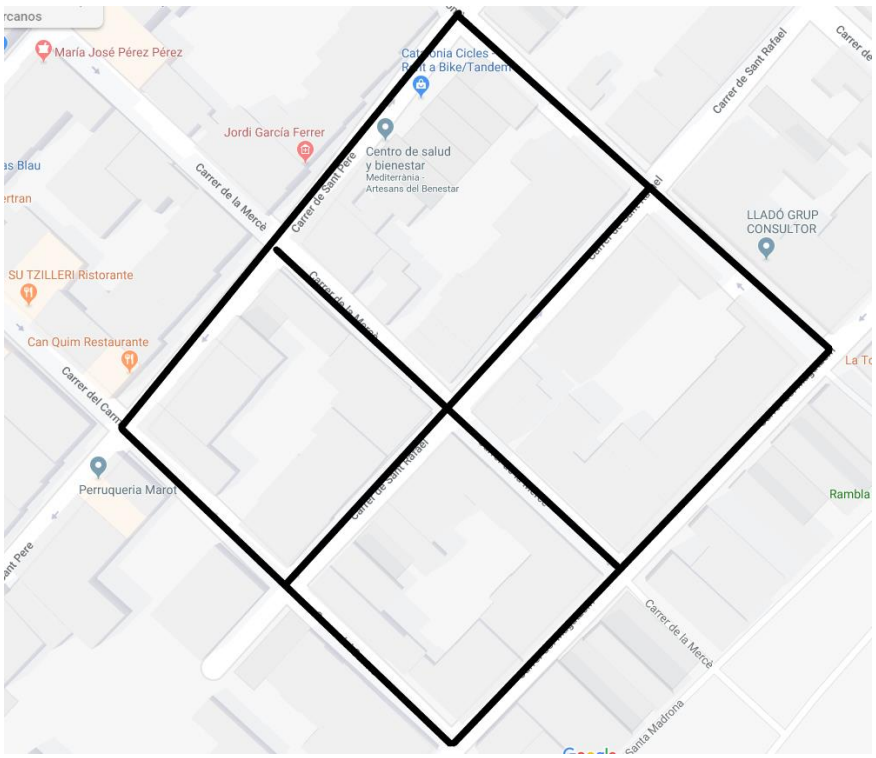

Ilustración 1 - Mapa del recorrido

1.3 Justificación

Con este proyecto se quiere demostrar que mediante el uso de tecnologías de bajo coste y con la utilización de software libre como pueden ser Processing o Arduino se puede realizar un proyecto muy versátil, ya que en este proyecto se hace un recorrido en bicicleta por ciudad, pero con la misma tecnología se podría hacer distintos recorridos, como puede ser una visita a un museo, recorrido en coche, por montaña etc.
Otro motivo más personal, aunque no tan importante, sería la de aprender con más profundidad a usar programas tan importantes de cara al mundo laboral como son Processing y sobre todo Arduino, ya que considero que, aunque hay asignaturas donde se usa el microcontrolador Arduino creo que no se hace con la profundidad suficiente, además de que solo se toca el hardware y no el software.

1.4 Alcance

Este proyecto incluye los siguientes puntos:

- Montaje de todos los componentes electrónicos, primero en una placa protoboard para comprobar el funcionamiento general y al final se hará la instalación es una bicicleta, para demostrar que el proyecto tiene un uso real.
- Programación del velocímetro en Arduino
- Comunicación vía puerto serie
- Programación del recorrido en Processing

En cambio no incluye:

- Diseño de una PCB para la instalación de todos los componentes
- Cálculo exacto de la velocidad en que se grabó el recorrido.
- Estudio de mercado.

1.5 Especificaciones Básicas

Las especificaciones básicas que ha de cumplir el sistema a desarrollar en este proyecto son las siguientes:

- Bajo coste del conjunto del sistema, por ese motivo se usara softwares libres y dispositivos de bajo coste.
- Fácil instalación de los distintos elementos.
- Utilizar la comunicación vía puerto serie para transmitir los datos.
- Control de la velocidad y dirección del recorrido en bicicleta.

1.6 Plan de trabajo

En primer lugar, se realizó un plan de trabajo que se tuvo que modificar debido algunas dificultades.
La programación de Processing, es decir crear un programa que me permitiera ejecutar múltiples videos, se alargó para que fuese totalmente funcional.
También hubo ciertas dificultades en la puesta en común del velocímetro de Arduino y el programa de Processing.
Debido a estos 2 problemas se necesitó más tiempo para realizar el proyecto.
Finalmente, les diferentes tascas se han organizado, desde septiembre de 2018 hasta mayo 2019, de la siguiente manera:

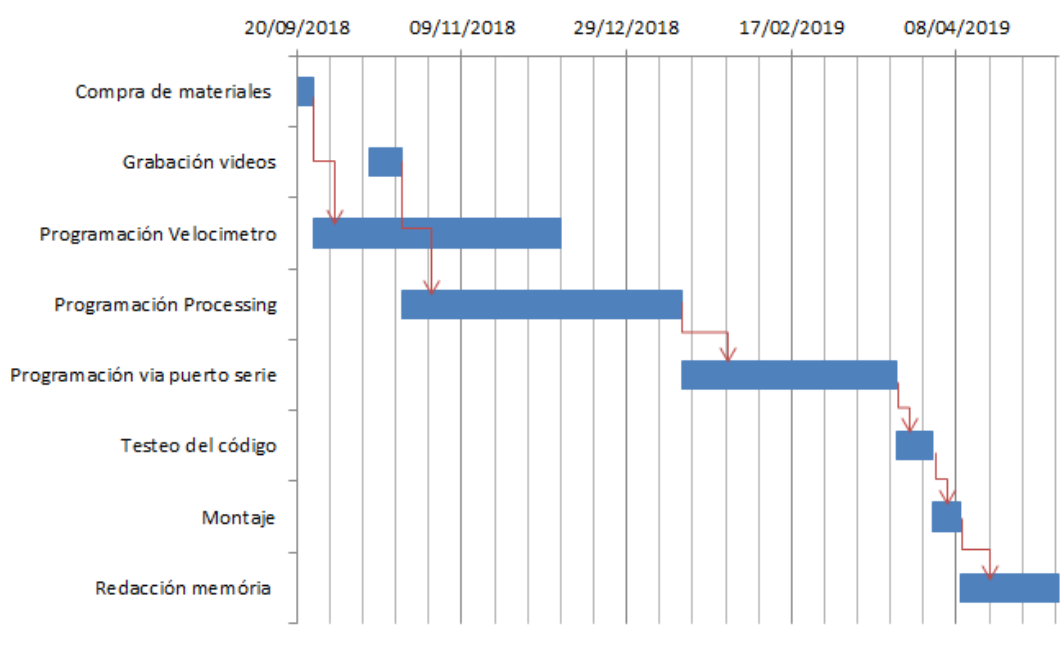

Ilustración 2 - Diagrama de Gantt

CAPÍTULO 2: SELECCIÓN DE LA TECNOLOGÍA

2.1 Comparativa de tecnologías

En este apartado se realizará una comparativa de las tecnologías usadas para llevar a cabo el proyecto "*Instalación audiovisual e interactiva*" y comentar el motivo por el que se han escogido este tipo de tecnologías por encima de las demás.
Estas son las diferentes opciones que servirían para llevar a cabo este proyecto, primero comentaremos los distintos softwares que se podrían haber usado y posteriormente el hardware.

- Software para crear la simulación del recorrido:
  - Processing
  - OpenFrameworks
  - Fugio

- Software para control del Microcontrolador:
  - Arduino
  - Eclipse Arduino IDE
  - PlatformIO

- Microcontrolador
  - Raspberry Pi
  - Arduino UNO R3
  - Arduino Mega

- Sensor velocidad
  - Sensor de efecto hall
  - Sensor de IQR

Los criterios aplicados para la selección del software han sido los siguientes:

- Facilidad de programación.
- Mayor comunidad en internet.
- Facilidad para la comunicación entre los dos softwares.

Y para el hardware los criterios de selección han sido los siguientes:

- Coste de implementación.
- Número de entradas y salidas digitales y analógicas, se necesita que el microcontrolador disponga de las suficientes entradas y salidas para poder llevar a cabo la instalación de todos los componentes.
- Reutilización del material, interesa que el material usado se pueda reutilizar en futuros proyectos.

Software

Características del software **Processing**:

- Rápido de aprender y escribir
- Considerado como un lenguaje en sí mismo
- Construido con su propio IDE
- Se vuelve lento cuando se usa para trabajo pesado.
- Implementa Java, que se creó después de C (y C ++). Java administra la memoria de la computadora automáticamente, y es por eso que puede ser lento.
- Se utiliza principalmente en visualización de datos, computación física, diseño algorítmico, aplicaciones interactivas simples y, lo más importante, educación.
- Muy amplia base de usuarios y libros.

Características del software **OpenFrameworks**: Dificultad moderada para aprender y escribir.

- Es un framework de C ++ para la codificación creativa.
- Alto rendimiento con gráficos 3D.
- Desde su C ++, permite al programador gestionar la memoria.
- Ampliamente utilizado en instalaciones interactivas, gráficos OpenGL y aplicaciones móviles iOS.

Características del software **Arduino[1]**:

- **Arduino tiene una gran comunidad:** Gracias a su gran alcance hay una gran comunidad trabajando con esta plataforma, lo cual genera una cantidad de documentación bastante extensa, la cual abarca casi cualquier necesidad.
- **Su entorno de programación es multiplataforma:** Se puede instalar y ejecutar en sistemas operativos Windows, Mac OS y Linux.
- **Lenguaje de programación de fácil compresión:** Su lenguaje de programación basado en C++ es de fácil compresión que permite una entrada sencilla a los nuevos programadores y a la vez con una capacidad tan grande, que los programadores más avanzados pueden exprimir todo el potencial de su lenguaje y adaptarlo a cualquier situación.
- **Re-usabilidad y versatilidad:** Es re-utilizable porque una vez terminado el proyecto es muy fácil poder desmontar los componentes externos a la placa y empezar con un nuevo proyecto, de igual manera todos los pines del microcontrolador están accesibles a través de conectores hembra, lo cual permite sacar partido de todas las bondades del microcontrolador con un riesgo muy bajo de hacer una conexión errónea.

Características del software **PlatformIO**: es una software basado en el Arduino IDE por lo tanto tiene sus mismas características pero te permite una mayor flexibilidad en comparación con Arduino, como contra es que es un software en crecimiento y aún tiene que pulir ciertos fallos y errores.

Respecto al software, la solución adoptada ha sido la de usar los programas de **Processing** y **Arduino** debido a su facilidad de aprendizaje así como la gran comunidad que hay y por ende la gran cantidad de información que se puede encontrar. Otro aspecto importante es que Arduino, es un programa IDE basado en Processing por lo tanto las interfaces y el lenguaje de programación de los dos programas son idénticos, con lo cual nos facilita mucho la programación, además de que la comunicación entre estos dos Programas vía puerto serie es muy sencilla.

Hardware

Para elegir el microcontrolador me he fijado sobretodo en la cantidad de entradas y salidas que proporciona y su coste. En el mercado hay una gran cantidad de microcontroladores Arduino distintos, eso es debido a que es un hardware libre y cada uno se puede crear uno a medida, pero de todos esto se ha elegido el Arduino UNO R3, aunque hay otros modelos como el Arduino Mega que dispone de más entradas y salidas tanto digitales como analógicas, para este proyecto con las que nos ofrece Arduino UNO R3 hay suficiente. Se podría haber hecho el proyecto con una Raspberry Pi pero esta está pensada para ser usada como un microordenador haciendo que sea más complejo de programar.

2.2 Solución Adoptada

Una vez escogida la solución final, los elementos necesarios para desarrollar el proyecto son los siguientes:

- Microcontrolador Arduino UNO R3 para poder realizar el velocímetro de la bicicleta y enviar los datos del velocímetro mediante puerto serie.
- Processing, programa utilizado para crear el recorrido virtual en bicicleta y para leer los datos enviados por Arduino mediante puerto serie, para así poder controlar distintos parámetros de la simulación del recorrido en bicicleta.
- Los componentes necesarios para realizar el proyecto son los siguientes:
    - Sensor de Efecto hall, para calcular la velocidad de la bicicleta, así como para calcular la distancia recorrida
    - 2 potenciómetros lineales de 10KΩ, uno para simular el giro del manillar, y el otro para controlar la luminosidad de la pantalla LCD
    - Pantalla LCD, donde se mostrará los valores de velocidad, distancia recorrida etc.
    - 3 leds, dos de ellos harán la función de indicador de giro y el otro para visualizar si el sensor de efecto hall está funcionando correctamente
    - 6 resistencias, 4 de 220Ω, 1 de 10KΩ y 1 de 5,1 KΩ las resistencias de 220Ω se usarán para controlar la intensidad que circula por los distintos leds mientras que las resistencias de 10 KΩ y 5,1 KΩ se usaran como resistencias Pull-Up, la de 10 KΩ irá con el sensor de efecto hall y la de 5,1 KΩ con el pulsador.
    - 2 pulsadores, uno para hacer un cambio de la información que se muestra por la pantalla LCD, y el otro pulsador para resetear el recorrido virtual en Processing.

A continuación tenemos un esquema (Ilustración 3) de la instalación de todos los componentes:

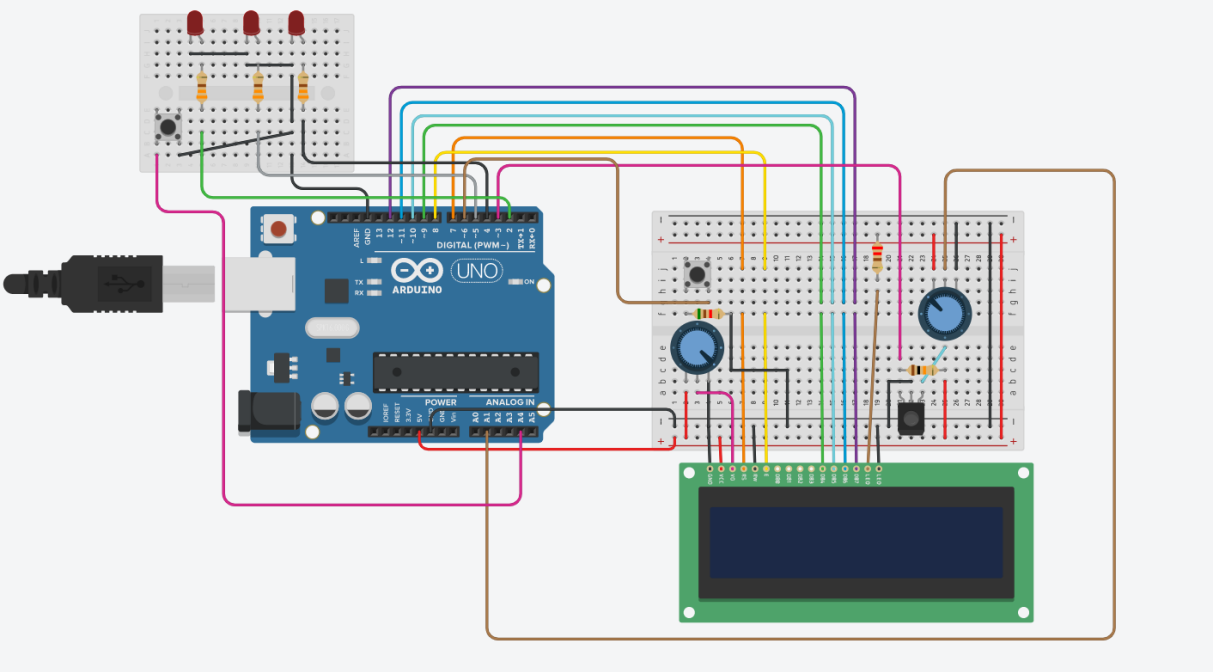

Ilustración 3 - Esquema eléctrico realizado en tinkercad

CAPÍTULO 3: Desarrollo

3.1 Descripción y especificación

En este apartado, explicare las funciones que tiene cada uno de los componentes y sus especificaciones.
El microcontrolador Arduino, como ya he explicado en el punto anterior, es el que se usará para almacenar los datos obtenidos del velocímetro y enviar estos datos mediante puerto serie hacia Processing, para este proyecto hemos usado el microcontrolador Arduino UNO R3 de elegoo.

3.1.1 Arduino UNO R3

Arduino UNO R3 (Ilustración 4) es un microcontrolador programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica [2]. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida [2]. Arduino se puede utilizar para desarrollar elementos autónomos, conectándose a dispositivos e interactuar tanto con el hardware como con el software. Nos sirve tanto para controlar un elemento como para leer la información de una fuente [3].

Ilustración 4 - Arduino UNO R3 de elegoo

Este dispositivo nos va permitir guardar los valores obtenidos por los sensores y los otros componentes y enviarlos a Processing.
En esta imagen (Ilustración 5) podemos ver todos los pines del microcontrolador Arduino y su función, así como en la tabla 1 se da información más detallada de la función de cada pin.

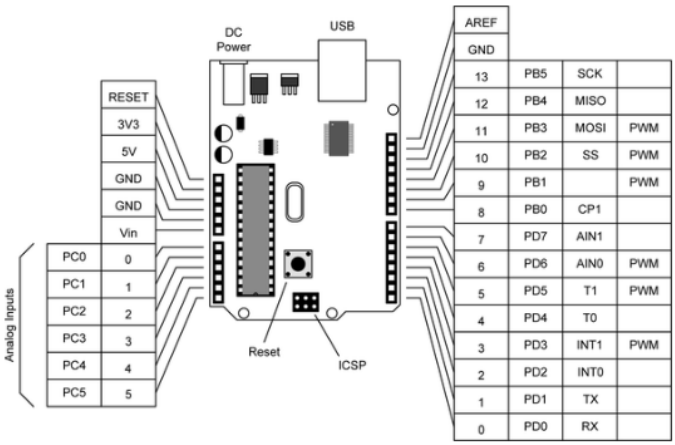

Ilustración 5 - Esquema de los pines de la placa Arduino[6]

Pines de alimentación	
Vin	Voltaje de entrada de la placa Arduino cuando se quiera usar una fuente de alimentación externa
5V	Este pin es una salida de la placa regulada a 5V. La paca se alimenta mediante la toma de corriente continuo (7-12V), por el conector USB (5V), no se puede usar los pines de 5V o 3,3V para alimentar la placa debido a que la placa se puede dañar ya que no pasan por ningún regulador de tensión.
3,3V	Pin que suministra 3,3V generados por el regulador de tensión de la placa. Corriente DC máximo 60mA
GND	Pin de tierra
IOREF	Pin de la placa Arduino que proporciona la referencia de tensión en la cual opera el microcontrolador
Entradas y salidas digitales	
Serie: 0(RX) y 1(TX)	Se utiliza para la recepción (RX) i transmisión (TX) de datos en serie TTL.
Interrupciones externas: 2 y3	Estos pines se pueden configurar para activar una interrupción, Arduino es capaz de detectar los siguientes eventos: - RISING, ocurre en el flanco de subida de LOW a HIGH. - FALLING, ocurre en el flanco de bajada de HIGH a LOW. - CHANGING, ocurre cuando el pin cambia de estado (rising + falling). - LOW, se ejecuta continuamente mientras está en estado LOW.
PWM:3,5,6,9,10,11	Las Salidas PWM (Pulse Width Modulation) permiten generar salidas analógicas desde pines digitales. Estos pines pueden proporcionar una salida PWM de 8 bits con la función analogWrite().

SCI: 10(SS), 11(MOSI), 12(MISO), 13(SCK)	Estos pines soportan la comunicación SPI, que, aunque proporcionada por el hardware subyacente, no se incluye actualmente en el lenguaje Arduino.
Led: 13	En el microcontrolador hay un LED incorporado al pin digital 13. Cuando el pin es de alto valor, el LED está encendido, cuando el pin es bajo, está apagado.
Entradas analógicas	
TWI: A4 o A5 i el pin SDA i SCL pin.	Pines que dan soporte a la comunicación I2C/TWI usando la librería wire
Otros pines	
AREF	Voltaje de referencia para las entradas analógicas, mediante la librería analogReference()
Reset	Solo en el microcontrolador Diecimila, en el caso de que su estado sea LOW se resetea el microcontrolador.

Tabla 1 - Función de los pines de la placa Arduino[5]

Las principales características técnicas del microcontrolador Arduino UNO R3 son las siguientes:

Microcontrolador	
Tipo	ATmega328P
Tensión de funcionamiento	5V
Voltaje de entrada(recomendado)	7-12V
voltaje de entrada(límite)	6-20V
Digital pines I/O(PWM)	14(de los cuales 6 proporcionan una salida
PWM digital pines I/O	6
Pines de entrada analógica	6
Corriente DC por Pin I/O	20mA
Corriente DC para Pin 3,3V	60mA
Memoria flash utilizados por el gestor	32KB ATmega328P de los que 0,5KB son arranque
SRAM	2KB ATmega328P
EEPROM	1KB ATmega328P
Velocidad de reloj	16Mhz
longitud	68,6mm
Anchura	53,4mm
Peso	25g

Tabla 2 - Características básicas del microcontrolador Arduino UNO R3 [4]

3.2 Componentes electrónicos.

En este apartado se presentarán todos los componentes electrónicos utilizados conjuntamente con el microcontrolador Arduino UNO R3, para llevar a cabo este proyecto.

3.2.1 Sensor de Efecto Hall

El sensor de efecto hall para la medición de campos magnéticos o corrientes o para la determinación de la posición en la que está. Encapsulado tipo SIP.
Su funcionamiento (Ilustración 6), es que al hacer circular una corriente eléctrica a lo largo de un semiconductor en presencia de un campo magnético, los electrones son desviados por efecto del campo magnético, dando lugar a una tensión perpendicular a la corriente y al campo magnético.

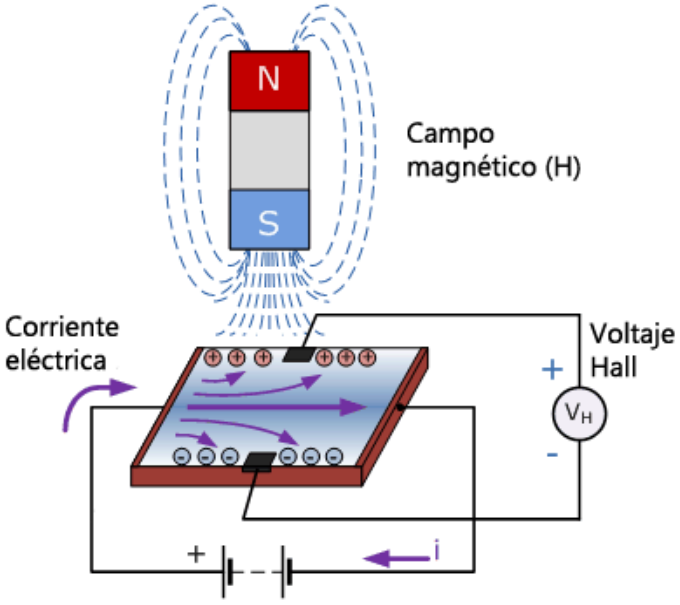

Ilustración 6 - Efecto Hall[9]

Generalmente nos encontramos con dos tipos de sensores Hall[9]:

- **Analógicos**. Generan una salida proporcional a la intensidad del campo magnético. Empleados para medir la intensidad de un campo magnético
- **Digitales**. Proporcionan un valor Alto en presencia de campo magnético, y bajo en ausencia del mismo. Por tanto, son empleados para detectar la existencia de campos magnéticos. A su vez se dividen en,
    - Switch, se activan al acercar el polo, y se desactivan al retirar el polo
    - Latch, se activan al acercar un polo, y mantienen su valor hasta que se acerca un polo contrario

En nuestro caso usaremos un sensor Hall A3144 (Ilustración 7), de tipo digital Switch.
Usaremos el sensor de efecto hall A3144 y un imán de neodimio para saber el número de vueltas de la rueda de la bicicleta y con estos datos calcularemos tanto la velocidad como la distancia recorrida.

Ilustración 7 - Sensor de efecto hall A3144

La instalación (Ilustración 8) de este sensor ira acompañado de una resistencia(10KΩ) de Pull-Up para establecer el estado HIGH cuando el sensor se encuentra en reposo, para así evitar los falsos estados que se producen por el ruido generado por los circuitos electrónicos.
Se usa una resistencia Pull-Up debido a que el sensor Hall A3144 tiene una configuración de salida de colector abierto, como consecuencia si conectas la salida directamente sin ninguna resistencia o con una resistencia Pull-Down, el resultado que obtendrás será siempre de 0V, por lo tanto la salida no se activaría nunca.

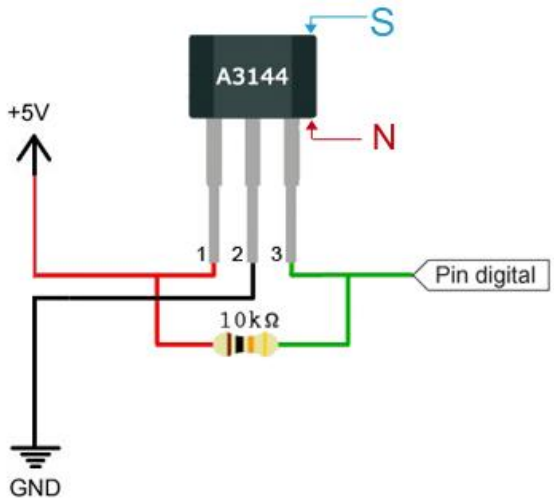

Ilustración 8 - Esquema eléctrico sensor efecto hall A3144[9]

A continuación (tabla 3) se pueden ver las características de este sensor.

Electrical characteristics at Vcc=8V						
Características	Symbol	Test Condition	Limits			
			Min	Typ	Max	Units
Supply Voltatge	VCC	Operating	4,5	-	24	V
Output Saturacion voltage	VOUT(SAT)	IOUT = 20 mA, B > BOP	-	175	400	mV
Output Leakage Current	IOFF	VOUT = 24 V, B < BRP	-	<1,0	10	µA
supply current	ICC	B < BRP (Output OFF)	-	4,4	9	mA
output rise time	tr	RL = 820 Ω, CL = 20 pF	-	0,04	2	µs
output fall time	tf	RL = 820 Ω, CL = 20 pF	-	0,18	2	µs

ABSOLUTE MAXIMUM RATINGS at TA = +25°C	
Supply voltatge	28
Reverse Battery Voltage, VRCC .	-35
Magnetic Flux Density, B	Unlimited
Output OFF Voltage, VOUT	28
Reverse Output Voltage, VOUT	-0,5
Continuous Output Current, IOUT	25mA
Operating Temperature Range, TA	
Suffix 'E–'	-40°C to +85°C
Suffix 'L–	-40°C to +150°C
Storage Temperature Range, Ts	-65°C to +170°C

Tabla 3 - Datasheets sensor efecto hall A3144[10]

3.2.2 Potenciómetro

Tanto para hacer la simulación de giro de la bicicleta como para controlar la luminosidad de la pantalla LCD se ha usado un potenciómetro rotativo (10kΩ) lineal. Se ha decantado por un potenciómetro lineal y no uno parabólico o exponencial debido a que los lineales presentan una proporcionalidad entre resistencia y desplazamiento, lo cual significa un comportamiento más intuitivo, pero no es tan preciso como los otros, pero en nuestro caso no se necesita un potenciómetro muy preciso.
A continuación vemos el esquema eléctrico (Ilustración 9).

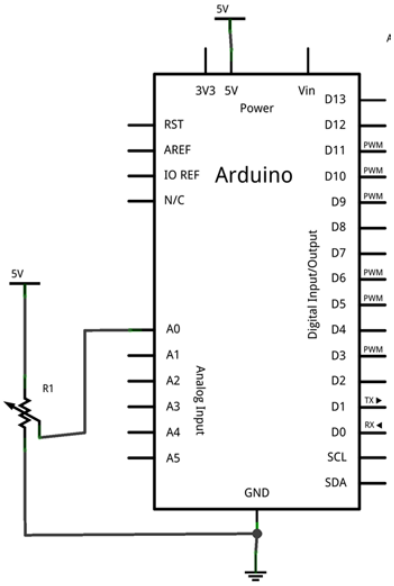

Ilustración 9 - Esquema eléctrico Potenciómetro[8]

3.2.3 Pantalla LCD 1602

LCD 1602 (Ilustración 10), la pantalla tiene una retroiluminación de led y puede mostrar dos filas con hasta 16 caracteres en cada fila, esta pantalla nos servirá para mostrar la información que nos proporciona el sensor de efecto hall de la velocidad, distancia recorrida, velocidad media y tiempo transcurrido desde el inicio del trayecto.

Ilustración 10 - Pantalla LCD 1602

La función de cada pin de la pantalla LCD es:

Pin	Función
VSS	Pin de tierra, GND
VDD	Pin de alimentación, 5V.
VO	Pin para regular mediante un potenciómetro de 10K el contraste de la pantalla.
RS	Pin de selección de registro, el microcontrolador le comunica al LCD si quiere mostrar caracteres o si lo que quiere es enviar comandos de control, como cambiar posición del cursor o borrar la pantalla etc.
R/W	Es el pin encargado de seleccionar el modo de escritura i lectura, para seleccionar el modo escritura de manera permanente hay que conectar el pin a GND.
E	Pin que habilita la pantalla para recibir información.
D0-D7	Pines para leer o escribir los datos recibidos. Donde D0 es el LSB (bit menos significativo) y el D7 MSB (el bit más significativo).
A y K	Pines para el control de la retroiluminación led.

Tabla 4 - Función pines LCD 1602[12]

3.2.3 Resistencias

Para este proyecto se ha usado resistencias de película de metálica(Ilustración 11) de 1/2W con una tolerancia del 1% Y coeficientes de temperatura bajos de hasta 50 ppm Hay 3 tamaños de carcasa disponibles: 0,25, 0,6, 0,75 W. La serie LR1L presenta un rango de valores óhmicos muy bajo de 0,1 a 0,82 ohmios. Especialmente idóneos en situaciones que requieren baja resistencia y tamaño reducido Las resistencias de película metálica tienen una excelente estabilidad bajo carga y en condiciones ambientales exigentes presentan bajos coeficientes de tensión y corriente de ruido.

Están disponibles en una amplia gama de valores de resistencia y son ideales para aplicaciones de precisión y uso general.

Las resistencias de 220 Ω se usaran para controlar la tensión que circula en los leds, en el pin V0 de la pantalla LCD y la resistencia de 10K Ω se usara como resistencia pull-up(explicado más arriba) del sensor de efecto hall.

Ilustración 11 - Resistencia con película metálica

Potencia a 70 °C	0,6 W
Tolerancia de resistencia	±0.5%
Coeficiente de temperatura	±50 ppm / °C
Temperaturas de funcionamiento	-55 °C → +155 °C
Tensión nominal de limitación	350 Vdc
Resistencia de aislamiento	220Ω Y 10kΩ
Tensión de sobrecarga:	700 Vdc (máx.)
Rigidez dieléctrica	700 Vdc
Longitud de cuerpo	6,2 mm
Diámetro de cuerpo	2,3 mm
Longitud del cable	25 mm
Diámetro del cable	0,55 mm

Tabla 5 - Características de las Resistencias[30]

3.2.4 Leds

Para este proyecto se han usado leds de la marca elegoo. Esta familia de LED consta de lámparas LED de 5 mm (T-1 3/4) de orificio pasante. Disponibles en una amplia variedad de colores, estos LED están disponibles con tipo de lente difuso o transparente.
Estos LED ofrecen un bajo consumo de potencia y una larga vida útil, y cuentan con una carcasa estándar industrial resistente y fiable.
Los leds se han usado como indicadores, el led azul se ha usado para comprobar el correcto funcionamiento del sensor de efecto hall, comprobando si detecta de manera correcta el imán. Los leds amarillos se han usado para indicar el valor del potenciómetro, para visualizar si el valor corresponde a un giro a la izquierda o a la derecha.

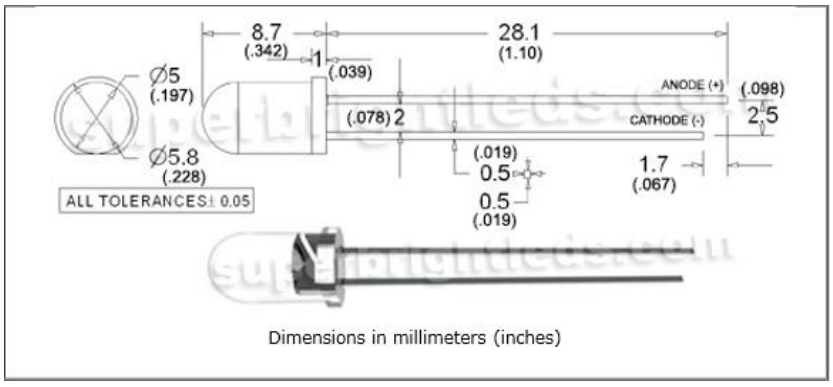

Ilustración 12- Dimensiones Led[29]

Parámetros	Símbolo	Valor	Unidad
Potencia de disipación	PD	150	mW
Corriente continua	IF	50	mA
Corriente de pico	IFP	100	mA
Tensión inversa	VR	5	V
Temperatura de funcionamiento	TA	-25~+80	°C
Temperatura de almacenamiento	TSTG	-40~+100	°C
Corriente inversa	IR	100	µA
Angulo de visión	2ø1/2	15	Degree
Tensión directa	VF	2	V
Longitud de onda de emisión máxima	ÿP	593	nm
Longitud de onda de emisión dominante	ÿD	590	nm
Línea espectral ancho medio	DI	15	nm
Intensidad luminosa	IV	56000	mcd

Tabla 6 - Datasheet led amarillo[30]

Parámetros	Símbolo	Valor	Unidad
Potencia de disipación	PD	120	mW
Corriente continua	IF	20	mA
Corriente de pico	IFP	50	mA
Tensión inversa	VR	5	V
Temperatura de funcionamiento	TA	-40~+85	°C
Temperatura de almacenamiento	TSTG	-40~+85	°C
Corriente inversa	IR	10	µA
Angulo de visión	2ø1/2	30	Degree
Tensión directa	VF	3,5	V
Longitud de onda de emisión dominante	ÿD	467	nm
Línea espectral ancho medio	DI	25,46	nm
Intensidad luminosa	IV	5500	mcd

Tabla 7 - Datasheet led azul[30]

3.2.5 Pulsador SPTD ON-MOM

Pulsador para PCB/ panel, que nos servirá para cambiar la información que sale por la pantalla LCD. Este pulsador tiene 3 pines, donde el pin central irá asociado el pin digital número 6 del microcontrolador Arduino UNO R3, y los otros 2 pines, uno ira a tierra y el otro se alimentara con 5V

Ilustración 13 - Pulsador SPTD

En la siguente tabla(8) se puede observar las características del pulsador.

Características	
Categoría	Pulsador
Fabricante	Nide copal Electronics
Serie	8N
estado del pulsador	Activo
Tipo	Estándar
Circuito	SPTD
Función de cambio	Momentánea, El interruptor o disyuntor vuelve a su posición original al quitar la carga
Valor de la corriente	6A(AC), 4A(DC)
Voltage AC	125V
Voltage DC	30V
Valor nominal del contacto	3 A a 250 V AC
Tipo de actuador	Émbolo
Dimensiones de corte del panel	Circular - 6.50mm Dia
Temperatura de operación	-30°C ~ 85°C
Vida útil	25000 ciclos

Tabla 8 - Características pulsador

3.2.5 Micro Pulsador

Micro pulsador que nos servirá para resetear el recorrido en bicicleta de Processing.
Para su instalación se ha usado una resistencia de Pull-down de 5KΩ, para evitar que haya fugas de corrientes hacia el pin del microcontrolador.

Ilustración 14 - Micro Pulsador 4 pines

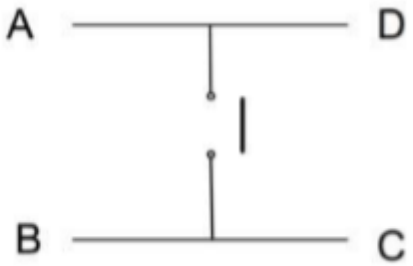

Ilustración 15 - Funcionamiento Pulsador[30]

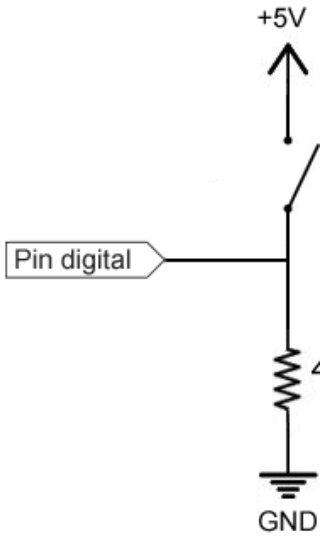

Ilustración 16 - Esquema eléctrico micro pulsador[24]

3.2.6 Imán de neodimio

Se trata de un imán permanente hecho de una aleación de neodimio (Ilustración 16), hierro y boro, combinados para formar un compuesto que cristaliza en el sistema cristalino tetragonal con la fórmula empírica $Nd_2Fe_{14}B$, como consecuencia posee una excepcional anisotropía magnética uniaxial[11].

El imán se colocara en el radio de la bicicleta, mientras que el sensor de efecto hall iría colocado en la vaina inferior, para así detectar el imán cada vez que la rueda gire.
Se ha elegido un imán de neodimio ya que al ser más potente que un imán normal genera un mayor campo magnético, asegurando así que el sensor lo detectará.

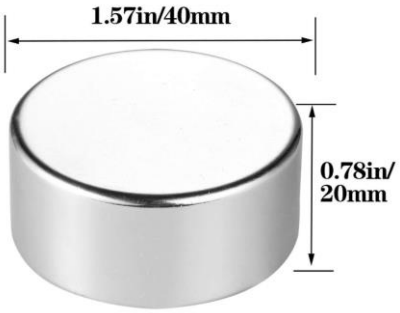

Ilustración 16 - Imán de neodimio

3.3 Montaje

Aquí se explicará cómo se ha realizado el montaje y el conexionado, se mostrará cómo se ha hecho la instalación sobre una placa protoboard y como se ha hecho esta instalación en una bicicleta para un uso real, de todos los componentes en el microcontrolador Arduino, situado en el manillar de la bicicleta. Antes de realizar el montaje en las placas de PCB, se realizó el montaje en una placa protoboard para comprobar el buen funcionamiento tanto del montaje como del programa, una vez comprobado que todo iba correctamente se realizó el montaje en el alojamiento.

Para realizar el montaje, primero de todo he soldado y cableado todos los componentes en la placa perforada PCB (Ilustración 17). A continuación se ha conectado los pines del microcontrolador Arduino a la placa PCB mediante "*Header Connectors*" (Ilustración 19).

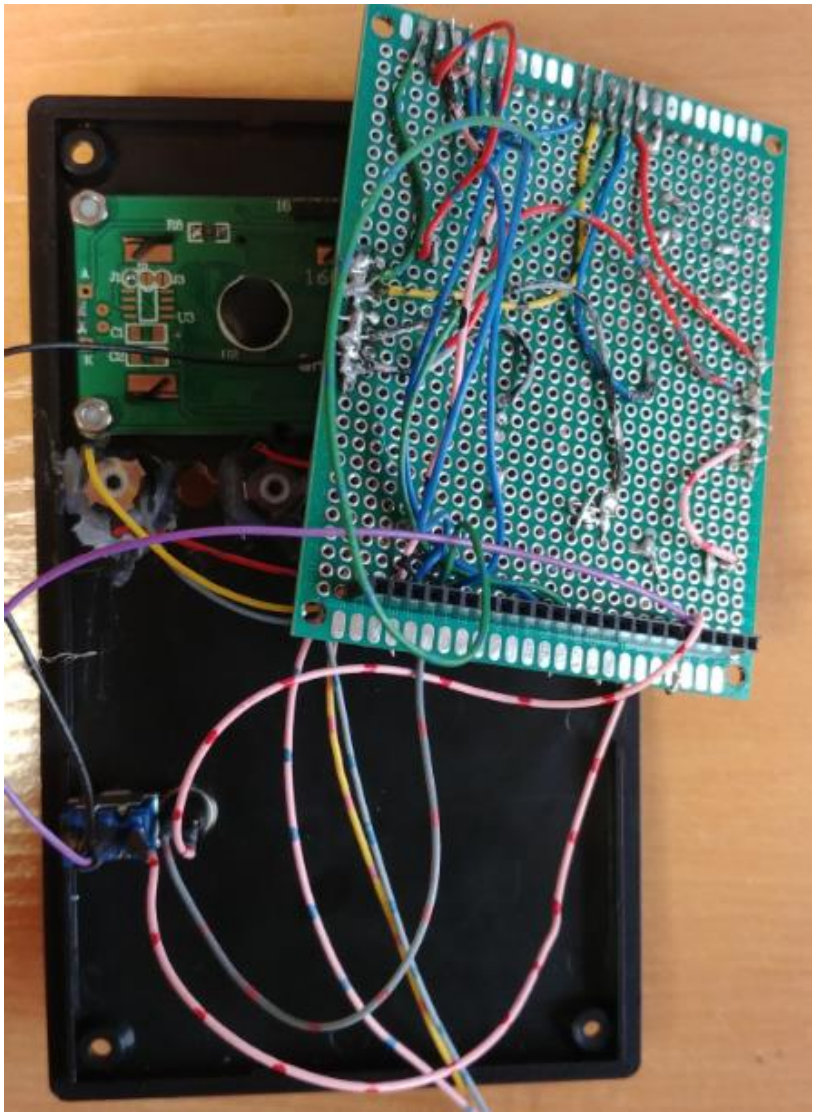

Ilustración 17 - Conexionado parte posterior de la placa PCB

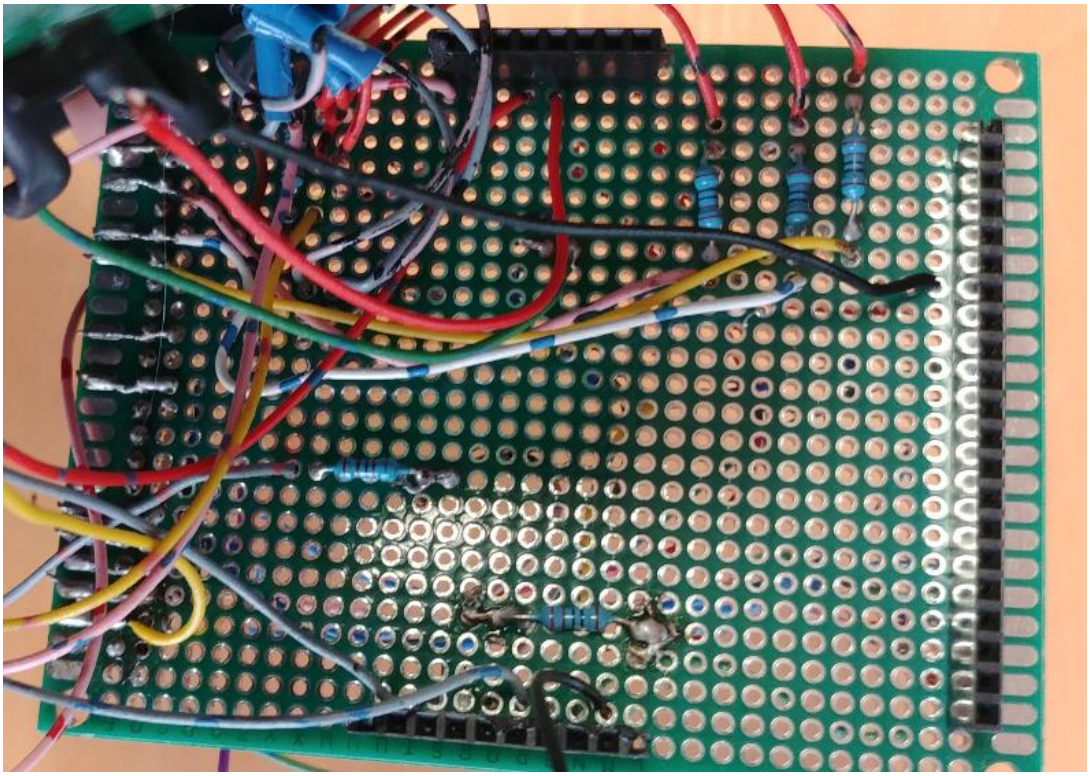

Ilustración 18 - Conexionado parte delantera de la placa PCB

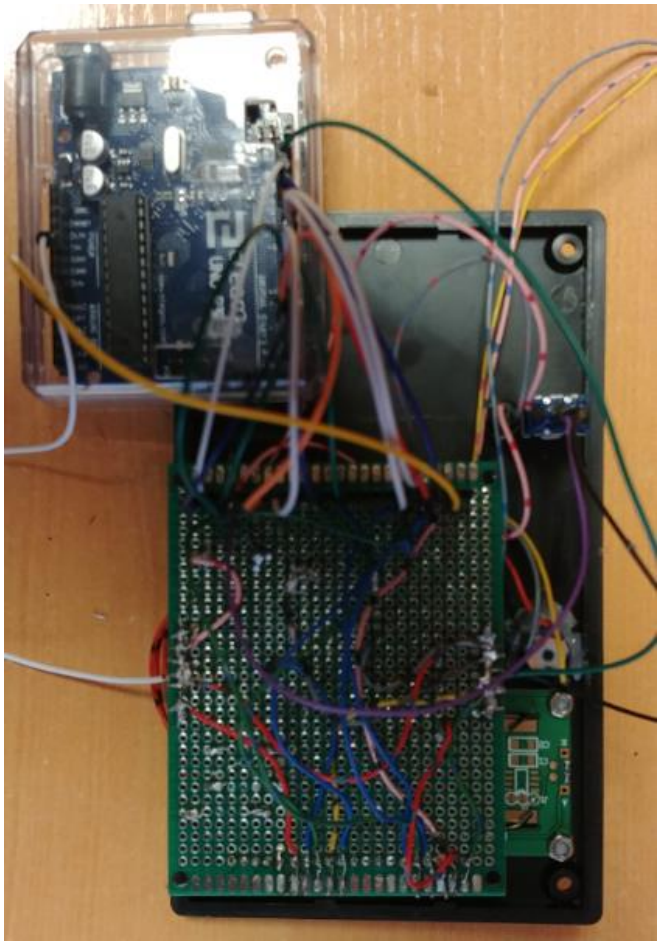

Ilustración 19 - Conexionado Arduino con placa PCB

Posteriormente, se ha creado un alojamiento (Ilustración 20) donde colocaremos tanto el microcontrolador Arduino UNO R3, como la placa perforada con todos los componentes instalados.

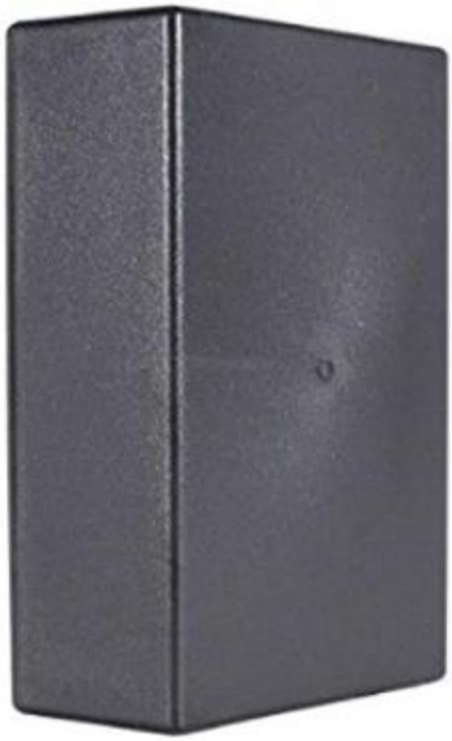

Ilustración 20 - Alojamiento de 160x95x55mm

Este alojamiento tendrá varios orificios (Ilustración 21 y 22) donde irán colocados la pantalla LCD, los leds, los pulsadores y los potenciómetros, para que así sean accesibles desde el exterior, además de tener tres orificios para facilitar la conexión del microcontrolador Arduino UNO R3 con otro periférico (ordenador en este caso) o con una fuente de tensión y otro que nos facilitará la conexión del sensor de efecto hall con la rueda de la bicicleta.

Ilustración 21 - Base del alojamiento con salida de USB y alimentación

Ilustración 22 - Tapa frontal del alojamiento

Finalmente en el alojamiento se le ha añadido un soporte para poder instalarlo en el manillar de la bicicleta (Ilustración 22).

Ilustración 23 - Soporte para el manillar de la bicicleta

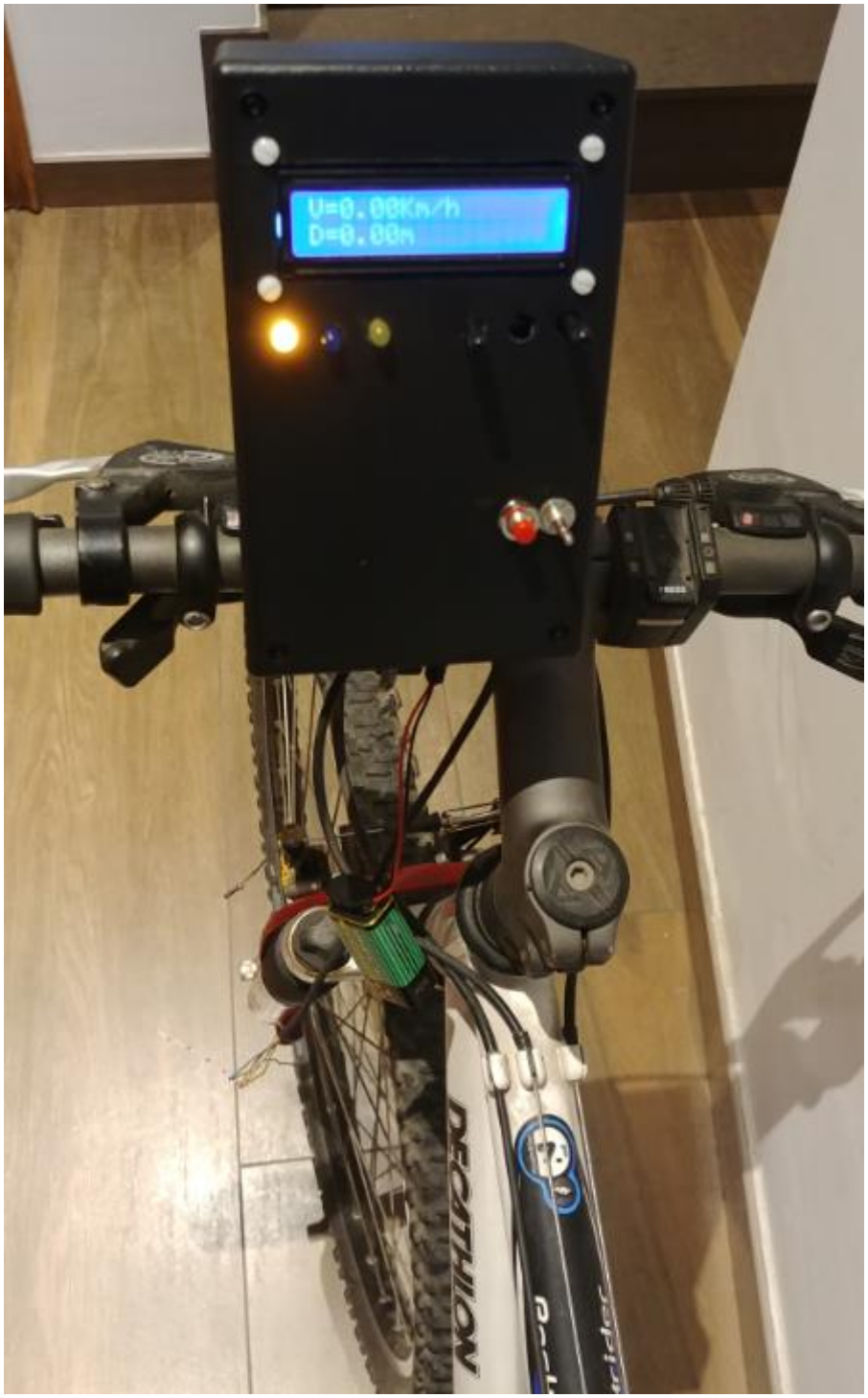

Ilustración 24 - Caja instalada en el manillar de la bicicleta

Ilustración 25 - Vista general del montaje en la bicicleta

3.4 Programación

En este apartado explicare las características principales del software usado en este proyecto:

- Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Cada esquema de Processing es en realidad una subclase de PApplet, un tipo Java que pone en funcionamiento la mayor parte de las características del lenguaje del Processing[15].

- El software usado para programar Arduino es un entorno de desarrollo integrado (IDE, basado en el entorno de *Processing* y en la estructura del lenguaje de programación Wiring). Es un programa informático compuesto por un conjunto de herramientas de programación.
  El IDE de Arduino es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Además incorpora las herramientas para cargar el programa ya compilado en la memoria flash del hardware[3].

Para realizar toda la programación de este proyecto, se ha usado Arduino para la programación y control del hardware, mientras que en Processing se ha usado para leer los datos enviados en Arduino mediante puerto serie para simular y controlar el recorrido en bicicleta.

3.4.1 Puerto Serie

Ahora explicare el funcionamiento de la interfaz de comunicación que usaré para intercambiar información entre Arduino y Processing.

El puerto serie envía información mediante una secuencia de bits, la información es transmitida bit a bit. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión) [19].

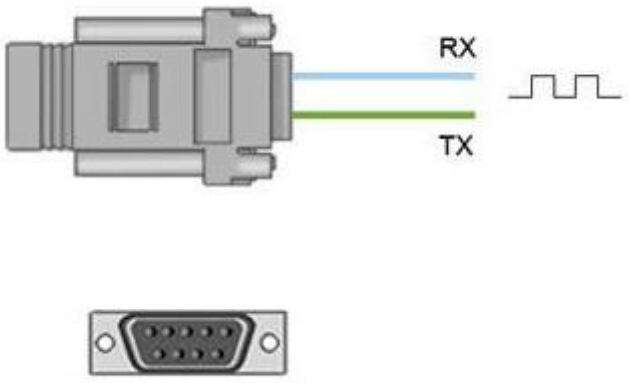

Ilustración 26 - Puerto Serie[19]

Un ordenador convencional dispone de varios puertos de serie. Los más conocidos son el popular USB (universal serial port). Sin embargo, dentro del ámbito de la informática y automatización existen una gran cantidad adicional de tipos de puertos serie, como por ejemplo el RS-485, I2C, SPI, Serial Ata, Pcie Express, Ethernet o FireWire, entre otros[19].

En ocasiones se puede referir a los puertos de serie como UART. La UART (universally asynchronous receiver/transmitter) es una unidad que incorporan ciertos procesadores, encargada de realizar la conversión de los datos a una secuencia de bits y transmitirlos o recibirlos a una velocidad determinada. Prácticamente todas las placas Arduino disponen al menos de una unidad UART. Las placa Arduino UNO R3 dispone de una unidad UART que opera a nivel TTL 0V / 5V, por lo que son directamente compatibles con la conexión USB[19].

Por otro lado, también está el término TTL (transistor-transistor logic). Esto significa que la comunicación se realiza mediante variaciones en la señal entre 0V y Vcc (donde Vcc suele ser 3.3V o 5V) [19].

Estos puertos están unidos físicamente a los pines del microcontrolador, en el Arduino UNO R3 los pines empleados son 0(RX) y 1(TX). En la siguiente imagen (Ilustración 27) se puede observar el diagrama de funcionamiento del puerto serie[19].

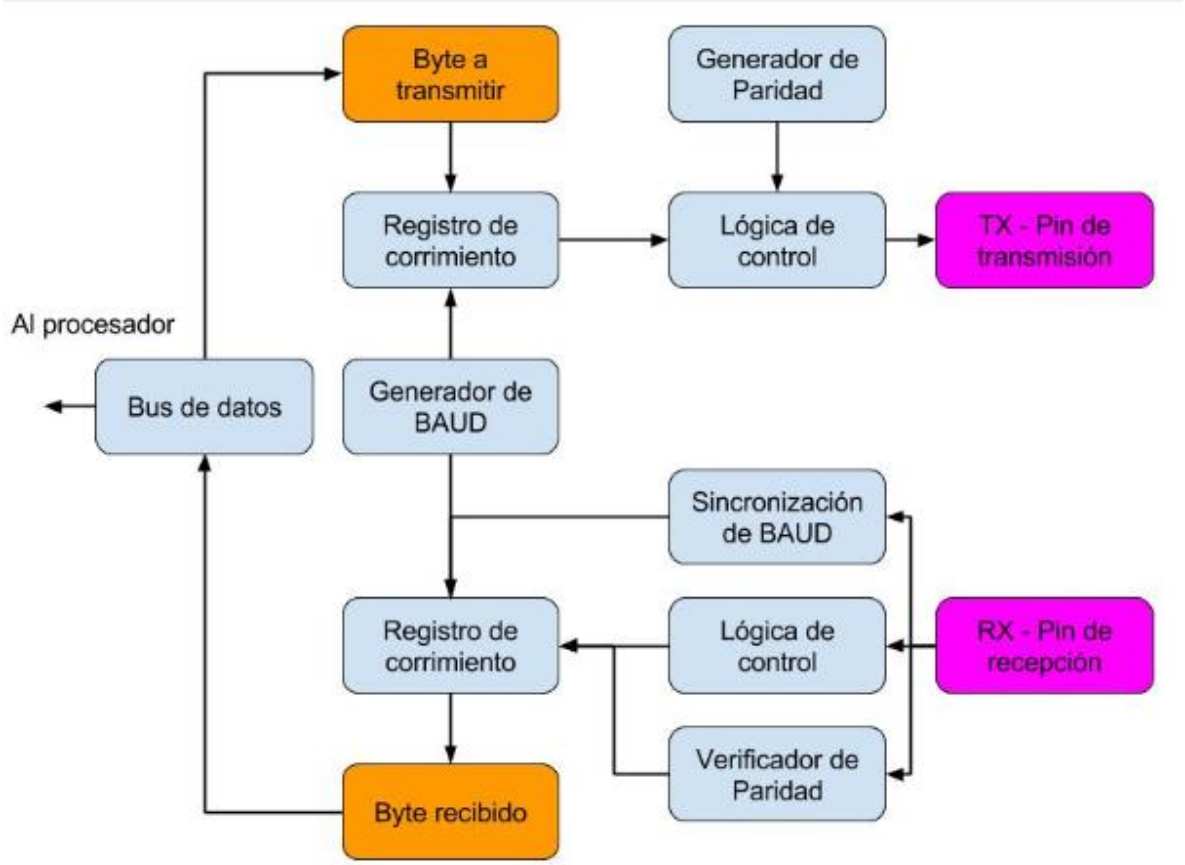

Ilustración 27 - Diagrama de un puerto serie[20]

Hay varias funciones para enviar o recibir datos por puerto serie como puede ser[7]:

- Serial-Write(): Escribe datos binarios en el puerto serie. Estos datos se envían como un byte o una serie de bytes.
- Serial.print(): Imprime datos en el puerto serie como texto ASCII legible. Este comando puede tomar muchas formas. Los números se imprimen utilizando un carácter ASCII para cada dígito. Los flotantes se imprimen de forma similar como dígitos ASCII, con dos decimales por defecto. Los bytes se envían como un solo carácter. Los caracteres y las cadenas se envían tal como están.
- Serial.println(): su función es la misma que Serial.print(), pero entre dato y dato deja un salto de línea.
- Serial.read(): Lee los datos seriales entrantes.
- Serial.readStringUntil(): lee los caracteres del búfer en serie en una cadena.

Y alguna función más, pero en este proyecto, se ha usado la función Serial.print() para enviar datos y Serial.read() para leer los datos que recibimos de Processing.

3.4.2 Arduino

En este archivo es donde se hace la programación, para realizar un velocímetro[14], que nos servirá para saber en todo momento la velocidad, la distancia recorrida, la velocidad media, el giro del manillar y el tiempo transcurrido. Toda esta información se mostrara por la pantalla LCD, además se enviara la información por puerto serie hacia Processing.
Primero de todo explicare las librerías que se han tenido que usar para poder hacer el programa:

```
#include <LiquidCrystal.h>
#include <timer.h>
```

- **LiquidCrystal.h:** Esta biblioteca permite que la placa Arduino controle pantallas LiquidCrystal (LCD) basadas en el chipset Hitachi HD44780 (u otra pantalla compatible), que se encuentra en la de las pantallas LCD basadas en texto. La biblioteca funciona en modo de 4 u 8bits(es decir, utilizando 4 u 8 líneas de datos además de la RS, habilitar y, opcionalmente, las líneas de control de R/W). Para crear una variable LiquidCrystal se hace de la siguiente manera:

  LiquidCrystal(rs, enable, d4, d5, d6, d7);

- **Timer.h:** Librería que nos permite crear temporizadores sin necesidad de usar la función delay(), ya que esta función pausa completamente el programa hasta que el tiempo fijado haya transcurrido. Con esta librería podremos crear timers que no afecten o detengan al programa y solo actuaran cuando el tiempo establecido haya transcurrido. Para crear una variable **Timer.h** se hace de la siguiente manera:

  Timer<1> default_timer;

  Donde el <n> es el número de eventos que se puede adjuntar a un temporizado, hasta un máximo de 10 eventos.
  A continuación se describirá el código que contiene el setup(), que es una parte del código que se ejecuta solo una vez y es cuando el programa se pone en marcha.

Setup()

Aquí podemos observar el código que contiene el setup(), que nos servirá principalmente para asociar cada pin del microcontrolador Arduino UNO con el componente correspondiente, indicar si el pin es de entrada o salida además de establecer la velocidad de transmisión, en 9600 baudios(bits por segundo), de datos por puerto serie.

```
void setup(){
   pinMode(giroIzqPin,OUTPUT);// Pin2.
   pinMode(giroDerPin,INPUT);// Pin5.
   pinMode(SensorHallPin,INPUT);// Pin 3
   pinMode(4,OUTPUT);// pin 4
   pinMode(A4,INPUT);
   Serial.begin(9600);
   lcd.begin(16,2);
```

Loop()

La función Loop() se repite de forma consecutiva lo que permite que su programa cambie y responda.

- Primero de todo calculamos el tiempo que ha transcurrido desde que se ha iniciado el programa, el tiempo se mostrará en segundos, pero una vez transcurridos los 60s se mostrará en minutos.
  La función millis() devuelve el número de milisegundos transcurridos desde que la placa Arduino comenzó a ejecutar el programa actual. Posteriormente en caso de que el pulsador de reset sea accionado, se enviara la variable "*Reset*" hacia Proccessing para reiniciar el recorrido virtual.

```
void loop(){
  TiempoTranscurrido = millis()/1000;

  if(TiempoTranscurrido >= 60){
    TiempoTranscurridoMin = millis()/60000;
  }

  if(digitalRead(PulsadorReset) == LOW && PulsacioResetAnterior == HIGH ){
    Reset = 1;
    Serial.print(Reset);
    Serial.print('\b');
  }

  PulsacioResetAnterior = digitalRead(PulsadorReset);
```

- Para comprobar que la rueda está realmente girando, guardamos el valor obtenido por el sensor de efecto hall en una variable y al cabo de 10ms vuelvo a leer este valor y lo guardo en otra variable, para saber que realmente lo que está detectando el sensor es el imán que hay en la rueda y no es una falsa lectura producida por el efecto bounce (efecto rebote).
  El efecto Bounce, es un efecto producido al ejecutar un conteo de vueltas, el sensor no reacciona o cuenta más de una vuelta.
  Esto se produce durante el primer milisegundo de cada conteo produciendo pequeñas variaciones de la señal de entrada que hacen que los valores HIGH y LOW obtenidos alternen rápidamente.

  Una vez hecho esto se podrá estar seguro de que la lectura del sensor es correcta por lo tanto cada vez que cambie de señal significará que la rueda de la bicicleta ha dado un vuelta y se incrementará el contador de vueltas y con esto podremos calcular la velocidad y la distancia recorrida.

```
EstadoActualSensor1=digitalRead(3);
delay(10);
EstadoActualSensor2=digitalRead(3);


        if (EstadoActualSensor1 == EstadoActualSensor2) {

            if (EstadoActualSensor1 != EstadoUltimoSensor){
                if (EstadoActualSensor1 == LOW) {
                    ContadorVueltas = ContadorVueltas + 1;
                    VEL();
                    Distancia();
                }

            }
    }
    EstadoUltimoSensor = EstadoActualSensor1;
```

- Obtenemos el valor del potenciómetro asociado al pin analógico A1 que nos servirá para simular el giro del manillar de la bicicleta.

```
TensionPote = analogRead(Analogpin);
PosicionPote =  map(TensionPote,0,1023,0,100);

if(PosicionPote != GiroAnterior){

    GiroAnterior=PosicionPote;
    Giro();
 }
```

Se puede observar como el valor que se obtiene de la lectura del potenciómetro viene dado en una resolución de 10bits, donde 0V=0 y 5V=1023, por lo tanto pasaremos este valor a tanto por ciento mediante la función map.
La función map(value, fromLow, fromHigh, toLow, toHigh) Re-mapea un número de un rango a otro. Es decir, un valor de fromLow se asignaría a toLow, un valor de fromHigh a toHigh, valores intermedios a valores intermedios, etc.
La comparación PosicionPote != GiroAnterior se hace para evitar que entre en el método constantemente y solo lo haga cuando la lectura del potenciómetro varíe.

- A continuación activamos el timer, mediante la función defaul_timer.tick(), para poder poner la velocidad a 0 en el caso de que la bicicleta lleve un rato detenida, ya que sin este timer la velocidad nunca se pondría a 0.

```
            Temporizador.tick();
            if (ContadorVueltas%2 == 0 ) {

                  digitalWrite(4, LOW);
                  ContadorParo1=0;
                  Temporizador.every(1000,Funcion2);

            }

            else {
                  digitalWrite(4, HIGH);
                  ContadorParo2=0;
                  Temporizador.every(1000, Funcion2);
            }

            PantallaLCD();

      }// Final del método loop()

      bool Funcion2(){

        ContadorParo2 = ContadorParo2 + 1;
        ContadorParo1 = ContadorParo1 + 1;

        if(ContadorParo2 == 3 or ContadorParo1 == 3){

          ContadorParo2=0;
          ContadorParo1=0;
          TimerAcabado2 = true;.
          VEL();

        }
        TimerAcabado2 = false;

      }
```

La idea de hacer esto es poner un timer cuando el contador de vueltas sea par o impar para asegurarse de que se resetea la velocidad, con los contadores de paro y la función defaul_timer.every(1000, funcion2).
Esta función lo que hace es que cada 1000ms llama al método funcion2. Una vez dentro del método lo que se hace es incrementar los contadores para que cuando uno de los 2 contadores llegue a tres, llamar al método VEL() para actualizar el valor de la velocidad.
Como se puede observar para que se actualice el valor de la velocidad los contadores tienen que llegar a tres, por lo tanto, como al método funcion2 se entra cada 1000ms, la velocidad se resteara cuando la bicicleta lleva 3s parada.

Métodos

En esta parte explicare la función que tiene cada uno de los métodos usados.

- void distancia(): En este método calcularemos la distancia recorrida por la bicicleta con los datos del contador de vueltas de la ruedo y con el perímetro de la rueda.
  La variable booleana esValorDistancia no servirá para el método floatToString. Este método se explicara más adelante.

```
void Distancia(){

   DistRecorrida=PerimetroRueda*ContadorVueltas;
   DistKM=DistRecorrida/1000;
   esValorDistancia = true;
   esValorVelocitat = false;.

   if(DistRecorrida <= 999){
     floatToString(DistRecorrida,6,2);
   }
   else {
     floatToString(DistKM,6,2);
   }

}
```

- void VEL(): Aquí calcularemos la velocidad de la bicicleta mediante las variables tiempo1 y tiempo2, cuando contador%2=0 (cuando el contador de vueltas sea par) se captura el tiempo actual y lo guarda en la variable tiempo1 y cuando el contador sea impar lo guarda en la variable tiempo2, con esto puedes obtener el lapso de tiempo que hay entre cada vuelta de la rueda(tiempo3), y finalmente con este tiempo pasado a horas y con el perímetro de la rueda calculas la velocidad.
  La velocidad la limitamos a un rango de 0 a 100Km/h.
  Después se convierte el valor de la velocidad de float a String mediante el método floatToString().

```
void VEL(){

        if (ContadorVueltas%2 == 0 ) {
              Tiempo1 = millis();

        }

        else {
              Tiempo2 = millis();
        }

        Tiempo3 = abs(Tiempo2-Tiempo1);

        Tiempo4 = (((Tiempo3/1000.0)/60)/60);

        Velocidad = ((PerimetroRueda/1000)/Tiempo4);

        if(Velocidad < 0){
           Velocidad = 0;
          }

          if(Velocidad >= 100){
            Velocidad = 100;
          }

          if(TimerAcabado2 == true){

            Velocidad = 0;
            ContadorParo1 = 0;
            ContadorParo2 = 0;
          }

          esValorDistancia = false;
          esValorVelocitat = true;
          floatToString(Velocidad,6,2);
          delay(10);
}
```

- void floatToString(float n, int i, int d): Con este método pasaremos los valores de distancia y velocidad de float a String, esta conversión hay que hacerla por dos motivos, uno es debido a que no es posible enviar datos del tipo float vía puerto serie mediante el método Serial.print(), y el otro motivo es que cuando tengamos que leer estos datos en Processing nos interesa poder diferenciar que dato estamos leyendo mediante la función Serial.readStringUntil().
  Los parámetros de este método tienen las siguientes funciones:
  - float n: es el número en formato float a convertir, en este caso la distancia o la velocidad.
  - Int i: Tamaño del número en caracteres.
  - Int d: precisión, el número de decimales que queremos obtener.
  - Char c/z: parámetro del método dtostrf, que es la cadena donde almacenaremos el número convertido, será un array con una longitud de i + 1.

  Finalmente se convierte el dato char en String mediante la función String(c).

Una vez convertido a String los valores de velocidad y distancia se podrán enviar a Processing vía puerto serie con la función Serial.print(), y para marcar el final de una transmisión enviaremos un carácter con el Serial.print("\t") o el Serial.print("\n"), para saber que dato se ha enviado y evitar errores en la lectura en Processing.

```
void floatToString(float n,int i,int d){
  if(esValorVelocitat==true){
    char z[i+1];
    String velreal;
    dtostrf(n,i,d,z);
    velreal=String(z);
    Serial.print(velreal);
    Serial.print("\t");
    delay(10);
  }

  if(esValorDistancia==true){
    char c[i+1];
    String Distancia;
    dtostrf(n,i,d,c);
    Distancia=String(c);
    Serial.print(Distancia);
    Serial.print("\n");
    delay(10);
  }

}
```

- Void giro(): método por donde enviaremos el valor del giro, obtenido a partir del potenciómetro, hacia Processing.
  La variable giro puedo tomar tres valores:
  - Giro = 0: cuando el rango del potenciómetro este entre el 33-66%, y servirá para indicar en el recorrido que se seguirá recto.
  - Giro = 1: cuando el rango del potenciómetro este entre el 0-33%, y servirá para indicar en el recorrido que girara hacia la izquierda.
  - Giro = 2: cuando el rango del potenciómetro este entre el 66-100%, y servirá para indicar en el recorrido que girara hacia la derecha.

  En este caso no es necesario pasar la variable de int a String, debido a que si se puede enviar datos int además de que en Processing el valor enviado se puede leer como un String.

```
void Giro(){

  if(PosicionPote >= 0 & PosicionPote <= 33){

     digitalWrite(PinGiroIzq,HIGH);
     digitalWrite(PinGiroDer,LOW);
     Gir=1;

     if(Gir != ValorGiroAnterior){

       ValorGiroAnterior = Gir;
       Serial.print(Gir);
       Serial.print("\r");
       delay(10);
    }
  }
```

- Void LCD(): Es donde se mostrará toda la información por la pantalla LCD, para poder hacerlo se tiene que hacer un cambio de pantalla , que se realizará mediante el pulsador asociado al pin digital 6 o bien con un pulsador que saldrá por pantalla en Processing y que cada vez que se pulse enviara una información hacia Arduino, que se leerá con la función Serial.read(), y la pantalla cambiará. Aquí se puede ver un ejemplo de la función de este método.

```
void PantallaLCD(){
 if(digitalRead(PulsadorLCD)== LOW && PulsacionAnterior == HIGH){

   lcd.clear();
   Screen=Screen+1;

 }
 PulsacionAnterior = digitalRead(PulsadorLCD);

 if (Serial.available() > 0) {
   State = Serial.read();

   if (State == '1') {

    lcd.clear();
    Screen = Screen+1;


   }
 }
 if(Screen == 4){
   Screen = 1;
   }


 if(Screen == 1){

   lcd.clear();
   lcd.write("V=");
   lcd.print(Velocidad);
   lcd.write("Km/h");
   lcd.setCursor(0,1);

   if(DistRecorrida<=999){

                              lcd.write("D=");
                              lcd.print(DistRecorrida);
                              lcd.write("m");
                        }

                        else{

                              lcd.write("D=");
                              lcd.print(DistKM);
                              lcd.write("Km");
                        }

 }
```

3.4.3 Processing

Este archivo es el que se encargará de ejecutar el recorrido en bicicleta que habré grabado anteriormente, y mediante los datos recibidos de Arduino, se podrá modificar la velocidad en que se reproduce el video, en que calle girar(dentro del recorrido) además de mostrar todos los valores, que también se muestran por el LCD, por pantalla.
Primero de todo explicare las bibliotecas que se han tenido que usar para poder desarrollar el programa:

```
import processing.video.Movie;
import org.gstreamer.elements.PlayBin2;
import processing.serial.*;
import interfascia.*;
```

- **Processing.video.Movie**: La biblioteca de vídeo reproduce archivos de película y captura datos de vídeo de una cámara. El vídeo puede capturarse desde cámaras USB, cámaras IEEE 1394 (FireWire) o tarjetas de vídeo con dispositivos de entrada compuestos o S-video conectados al ordenador. Las películas se pueden cargar desde archivos ubicados en su ordenador o en cualquier lugar de Internet. Se basa en el marco multimedia GStreamer y utiliza los enlaces GStreamer-Java para interconectar GStreamer desde Java, que permite soportar una amplia gama de formatos de video (mp4, mov etc.).
  Gracias a esta biblioteca nos permitirá usar la clase Movie (), esta clase carga películas y las reproduce de muchas maneras, incluido el bucle, la pausa y el cambio de velocidad.

- **Org.gstreamer.elements.PlayBin2**: GStreamer es una biblioteca para construir gráficos de componentes de manejo de medios. Las aplicaciones que admite van desde la simple reproducción Ogg / Vorbis, la transmisión de audio / video hasta el procesamiento complejo de audio (mezcla) y video (edición no lineal).
  GStreamer ya incluye componentes para crear un reproductor multimedia que puede admitir una gran variedad de formatos, incluidos MP3, Ogg / Vorbis, MPEG-1/2, AVI, Quicktime, mod y más. GStreamer, sin embargo, es mucho más que otro reproductor multimedia. Sus principales ventajas son que los componentes conectables se pueden mezclar y combinar en tuberías arbitrarias para que sea posible escribir una aplicación de edición de audio o video.
  Playbin proporciona una abstracción completa de todo en uno para un reproductor de audio y / o video. Playbin puede manejar archivos de audio y video con las siguientes características:
  - reconocimiento automático del tipo de archivo y basado en esa selección y uso automáticos de los demuxers / decodificadores de audio / video / subtítulos correctos
  - Visualización de archivos de audio.
  - Soporte de subtítulos para archivos de video. Los subtítulos se pueden almacenar en archivos externos
  - Selección de transmisión entre diferentes transmisiones de video / audio / subtítulos
  - extracción de meta información (etiqueta)
  - Fácil acceso a la última muestra de video.
  - almacenamiento en búfer al reproducir transmisiones a través de una red
  - Control de volumen con opción de silencio

- **Processing.serial**: La biblioteca Serial lee y escribe datos en y desde dispositivos externos, solo un byte a la vez. Permite que dos computadoras envíen y reciban datos. Esta biblioteca tiene la flexibilidad de comunicarse con dispositivos como microcontroladores personalizados y usarlos como entrada o salida para los programas de Processing. El puerto serie es un puerto de E / S de nueve pines que existe en muchas PC y se puede emular a través de USB.
  Esta biblioteca es la que nos va permitir comunicar Processing y Arduino para que se puedan enviar entre ellos datos vía puerto serie.

- **Interfascia**: Es una biblioteca de interfaz gráfica de usuario para el entorno de programación de gráficos de procesamiento. Proporciona un conjunto de herramientas de widgets de interfaz estándar como campos de texto, botones, casillas de verificación, controles deslizantes, etc. Interfascia maneja automáticamente las interacciones dentro de la colección de widgets y envía mensajes de eventos a su proyecto. Podremos crear un botón, que se muestra por pantalla, que nos sirva para cambiar la información que hay por pantalla, además de enviar un dato en formato int hacia Arduino para así cambiar también la información que se muestra la pantalla LCD.

Una vez explicado las características de las bibliotecas usadas en este proyecto comentaremos la función de cada método usado en el programa.

Setup()

Al igual que Arduino el método setup() es una parte del código que se ejecuta solo una vez y es cuando el programa se pone en marcha y sirve para inicializar variables y crear la ventana principal por donde se mostrara todo.

```
void setup() {
  size(1280, 800, P2D);
  noSmooth();
  frameRate(FPS);
  background(0);
  j2 = createGraphics(width/4, height/10, P2D);
  informacion = new GUIController(this);
  b1 = new IFButton("Informacion",100,600,width/4, height/20);
  b1.addActionListener(this);
  informacion.add(b1);
  Texto = createFont("Arial",22,true);
  String portName = Serial.list()[1];
  myPort = new Serial(this, portName, 9600);
  for (String s : FILMS)  (movies[idx++] = new Movie(this, s)).playbin.connect(FINISHING);
  CarrerMerce();
  noLoop();

}
```

A continuación se explica las diferentes instrucciones y variables del código anterior.

- **Size(width, height, renderer)**: Define la dimensión del ancho y el alto de la ventana de visualización en unidades de píxeles. En un programa que tiene la función de setup(), la función de size() debe ser

la primera línea de código dentro de la setup(). El tercer parámetro es para especificar el motor de renderización que se va usar. Por ejemplo, si va a dibujar formas en 3D, use P3D. Además del renderizador predeterminado, hoy otros renderizadores:

- P2D (Processing 2D): Procesador de gráficos 2D que utiliza hardware de gráficos compatible con OpenGL.
- P3D (Processing 3D): Procesador de gráficos 3D que hace uso de hardware de gráficos compatible con OpenGL.
- FX2D (JavaFX 2D): Un renderizador 2D que usa JavaFX, que puede ser más rápido para algunas aplicaciones, pero tiene algunas peculiaridades de compatibilidad.
- PDF: El procesador de PDF dibuja gráficos 2D directamente en un archivo PDF de Acrobat.
- SVG: El procesador SVG dibuja gráficos 2D directamente en un archivo SVG.

En este caso nos va interesar usar el renderizador P2D.

- **Smooth()**;Dibuja toda la geometría y las fuentes con bordes irregulares (aliased) e imágenes cuando los bordes duros entre los píxeles se amplían, en lugar de interpolar los píxeles.

- **noLoop()**:Detiene el procesamiento de la ejecución continua del código dentro de draw (). Si se usa noLoop () en setup (), debe ser la última línea dentro del bloque. Este método se usa para detener el método draw().

- **Framerate()**:Especifica el número de frames que se mostrarán cada segundo. En este caso la variable FPS tiene un valor de 30 por lo tanto se ha fijado que los videos irán a una tasa de 30FPS.

- **Background**: Con esta función establecemos el color utilizado para el fondo de la pantalla principal de Processing. Siendo el 0 para el color negro y el 255 para el blanco.

- **J2**: Variable tipo PGraphics, para crear una nueva ventana dentro de la pantalla principal. Se crea un nuevo objeto PGraphics con los parámetros de width, height, renderer. Esta nueva pantalla es donde se mostrará la información de la velocidad, distancia recorrida, tiempo transcurrido y velocidad media.

- **B1:** Objeto del tipo IfButton (de la biblioteca interfascia) que nos servirá para cambiar la información que sale en la ventana j2, se le añade una actionListenner para que cuando se accione el botón se llame al evento void actionPerformed(GUIEvent e), que se explicara más adelante.

- **Información**: Se crea un nuevo objeto tipo GUIController. El objeto GUIController realiza un seguimiento de todos los componentes de la GUI y maneja el reenvío de eventos desde el applet de procesamiento a los componentes de la GUI individuales, es decir nos servirá para llamar a eventos en caso de que una acción suceda en otro componente GUI. Como se puede observar en el código, en el objeto informacion se le añade el objeto IfButton b1, para que así cada vez que se accione el pulsador, se llamará al evento, void actionPerformed(GUIEvent e), para realizar la acción correspondiente.

- **Texto**: Variable PFont que servirá para crear una formato de texto, en este caso formato Arial, tamaño de letra 22.

- **myPort**: Creas un nuevo puerto para permitir la comunicación por puerto serie y estableces una velocidad de comunicación de 9600baudios, es importante que la velocidad de comunicación coincida con la elegida en Arduino.

Draw()

Este método al igual que el método loop() de Arduino se repite de forma consecutiva lo que permite que su programa cambie y responda.
A continuación explicaremos las partes más importante del código que se halla dentro del método draw.

- Primero de todo fijamos la posición en la pantalla donde se mostraran los videos (m) mediante la función set(), las variables x, y definen las coordenadas de la esquina superior izquierda de la pantalla en este caso, como el valor de las variables es cero, se empezará por las coordenadas 0,0. Después se llamará al método "*drawRect(j2)*" donde se creará un rectángulo, en el cual se mostrará toda la información, que se situará dentro de la ventana principal en las coordenadas 100,650.

```
set(x, y, m);
drawRect(j2);
image(j2, 100, 650);
```

- A continuación procederemos a leer los datos enviados de Arduino por puerto serie.
  Para leer los datos recibidos, donde los valores enviados son de tipo String, primero de todo comprobamos que el puerto está disponible, y después usaremos la función *myPort.readStringUntil('\t')*, que con esto conseguiremos diferenciar el valor que se quiere leer ya que solo leerá los datos que al final de la cadena contengan el carácter '\t'.
  Nos interesa pasar el valor recibido de String a float y para esto usaremos la función *float(trim(ValorVelocidad))* donde *trim* tiene la función de eliminar los caracteres de espacio en blanco del principio y final de una cadena, es decir elimina el carácter(\t'). Esto se repetirá para la lectura de la distancia y del valor de giro.

```
if(myPort.available() > 0){

   delay(10);
   TransmisionVelocidad=myPort.readStringUntil('\t');

   if(TransmisionVelocidad!=null){
     ValorVelocidad=TransmisionVelocidad;
     FloatVelocidad= float(trim(ValorVelocidad));


   }
```

- Finalmente la llamada a los métodos donde se ejecutaran los videos se realizará de la siguiente manera:

```
if(carrermerce==true){
  CarrerMerce();
}
else if(mercerecta==true){
  Mercerecta();
}
 else if(rafaelizq==true){
  SantRafaelizq();
}
else if(santrafaeldreta==true){
  SantRafaeldreta();
}
```

Se ha decidido aplicar este método, debido a un problema que se ha tenido, que es cuando se tiene que comprobar se una video había terminado, mediante el evento de final de video final PlayBin2.ABOUT_TO_FINISH FINISHING = new PlayBin2.ABOUT_TO_FINISH() , dentro de los métodos donde se ejecutan los videos, si no entras constantemente, en bucle, a estos métodos lo comprobación no daba buen resultado.

Métodos

En esta apartado explicaré los distintos métodos usados.

- Métodos donde se ejecutarán los videos, en la siguiente imagen se puede observan un ejemplo.
  Lo primero que se puede observar, es que la variable "Reset" se le asigna el valor 0, para que así solo se resetee una vez.
  Posteriormente hay una variable booleana "*carrermerce*" donde se fija su valor a true, esto es debido a que si no se hace de esta manera, en este método solo se entraría una vez, y por lo tanto cuando se haga la comparación de si el video a terminado el resultado siempre seria que NO, por eso se pone esta variable, para entrar en una especie de bucle hasta que el video termine, seguidamente se puede observar que se ejecuta el video 0 y llamamos el método "*centralizeMovie()*" para centrar el video en la pantalla, después fijamos el valor de velocidad a un mínimo de 0.1 para simular que la bicicleta esta parada cuando se detecta que la velocidad es 0, en caso contrario la velocidad de reproducción será igual al valor de la velocidad obtenida de Arduino entre la velocidad en que se grabó el video.
  Después se comprueba si el video a terminado con la variable "*VideoTerminado*", que se obtiene del evento "*final PlayBin2.ABOUT_TO_FINISH FINISHING*", que se comentara más adelante, una vez el video ha terminado ponemos la variable "*carrermerce*" a false para salir del bucle, y saltamos al siguiente método, donde se ejecutara otro video, según el valor de "Giro" que se ha obtenido de Arduino.

```
void CarrerMerce(){

  Reset = 0;
  carrermerce=true;
  (m = movies[idx = 0]).play();
  while (m.width == 0)  delay(100);
  centralizeMovie();

  if(FloatVelocidad==null || FloatVelocidad<= 0 ){
    VelocidaddeRepoduccion=0.1;
  }

  else{
    VelocidaddeRepoduccion=FloatVelocidad/10.89;
  }

  m.speed(VelocidaddeRepoduccion);


  if(VideoTerminado == true){

      carrermerce=false;
      VideoTerminado=false;
      switch(Giro){
        case 0:
            m.stop();
            delay(100);
            Mercerecta();
            break;

          case 1:
            m.stop();
            delay(100);
            SantRafaelizq();
            break;

          case 2:
            m.stop();
            delay(100);
            SantRafaeldreta();
            break;

   }
  }

}
```

- **void actionPerformed(GUIEvent e)**: Evento que se llama cada vez que el pulsador b1 es accionado, para incrementar en 1 el valor de la variable "*screen*" que es la variable que hará cambiar la información que sale por la pantalla "*j2*", además se envía por puerto serie el valor de "1" cada vez que se pulsa el botón para así cambiar también la información que se muestra por el LCD.

```
void actionPerformed(GUIEvent e){
  if(e.getSource()==b1){
    screen= screen+1;
    myPort.write('1');
      if(screen == 3){
        screen=1;

      }
  }
}
```

- **void drawRect(PGraphics pg)**: En este método es donde mostraremos toda la información por el rectángulo creado.
  En esta imagen se pude observar una parte del código de cómo crear el rectángulo y añadir la información dentro de él.

```
void drawRect(PGraphics pg){
  pg.beginDraw();
  pg.clear();
  pg.stroke(128);
  pg.strokeWeight(10);
  pg.fill(128, 55, 0, 128);
  pg.rect(0, 0, pg.width, pg.height);



if(screen == 1){

      pg.textFont(Texto);
      pg.fill(0,102,0);
      pg.textAlign(CENTER);
      pg.text("Velocidad = "+FloatVelocidad+"Km/h",pg.width/2,pg.height/2.25);
      if(FloatDistancia<=999){
        pg.text("Distancia = "+FloatDistancia+"m",pg.width/2,pg.height/1.5);

      }

      else{

        pg.text("Distancia = "+FloatDistancia+"Km",pg.width/2,pg.height/1.5);

      }

}
```

Ahora procederé a explicar las funciones que se han usado.

- beginDraw(): Esta función establece las propiedades predeterminadas de un objeto PGraphics. Debe llamarse al principio de todo, antes de dibujar algo en el objeto.
- Clear():Borra los píxeles dentro de un búfer.
- Stroke():Establece el color utilizado para dibujar líneas y bordes alrededor del rectángulo.
- StrokeWeight(): Establece el grosor que tendrá el borde del rectángulo.
- Fill():Función para establecer el color de relleno, en RGB.
- Rect():Dibuja un rectángulo que tendrá el tamaño especificado en el objeto PGraphics "*j2*".
- TextFont(): Escogemos el tipo de letra del objeto tipo PFont "*Texto*" explicado anteriormente.
- textAlign():Establece la alineación actual para el texto de dibujo, en este caso en el centro del rectángulo.
- Text(): Función que sirve para escribir el texto que se quiere mostrar. Los dos últimos parámetros de esta función son las coordenadas "x" e "y" del texto.

- Void movieEvent(Movie m): Evento que se ejecuta cada vez que se reproduce un video nuevo. Dentro de este evento, leeremos el video que se quiere ejecutar y se actualizará la ventana principal para que se muestre el nuevo video.

```
void movieEvent(Movie m) {
  m.read();
  redraw();
}
```

- Void centralizeMovie(): método para controlar que el video siempre salga centrado y partiendo de las coordenadas (0, 0).

```
void centralizeMovie() {
  x = width  - m.width  >> 1;
  y = height - m.height >> 1;
}
```

- final PlayBin2.ABOUT_TO_FINISH FINISHING: Evento que se activa cada vez que un video termina. Este evento servirá para actualiza el valor de la variable "*VideoTerminado*", además de imprimir por consola el nombre del video que acaba de terminar.

CAPÍTULO 4: PRESSUPUESTO

En este capítulo se presentarán todos los costes relacionados al hardware y software que se ha necesitado para desarrollar este proyecto.

Además de los costes de materiales, se calculara el coste para una empresa realizar este proyecto en el supuesto de que la persona que lo ha realizado sea un ingeniero junior.
Un ingeniero junior es aquel que se inicia en el mundo laboral y aún está en proceso de aprendizaje.

Para este presupuesto se he enfocado desde el punto de vista de un ingeniero junior que trabaja en una empresa, con un sueldo bruto mensual de 1300€ (en el mercado laboral actual el sueldo de un ingeniero junior oscila entre 1200 – 1800€ bruto mes).

4.1 Coste del velocímetro

Los costes de los diferentes materiales usados en el desarrollo del velocímetro son presentados en este apartado.

Nombre	Descripción	Cantidad	Precio Unitario[€]	Coste total[€]
Arduino UNO R3	Microcontrolador ATmega328P con 14 I/O digitales y 6 entradas analógicas	1	5.40	5.40
Potenciómetro	Potenciómetro lineal de 10KΩ	2	0.85	1.70
Pantalla LCD	Pantalla LCD 1602	1	5.50	5.50
Resistencias	Resistencia de 220Ω	4	0.16	0.64
Resistencia	Resistencia de 10KΩ	1	0.159	0.159
Resistencia	Resistencia de 5,1KΩ	1	0.42	0.42
Led	Leds de color amarillo	1(caja)	7.99	7.99
Sensor de efecto hall	Sensor A3144, Vcc=4 a 24V	1	0.85	0.85
imán	Imán de neodimio	1	12	12
Pulsador	Pulsador SPTC de acción momentánea	1	19.12	19.12

Micro Pulsador	Pulsador de cuatro pines	1	0.989	0.989
Subtotal				54.77€

Tabla 9 - Coste del velocímetro

4.2 Coste del montaje

Coste de los materiales y herramientas necesarias para realizar el montaje

Nombre	Descripción	Cantidad	Precio Unitario[€]	Coste total[€]
Caja para Arduino UNO R3	Caja de protección para Arduino	1	5.99	5.99
Placa PCB	Placas PCB de doble cara de 7x9cm	1	2.89	0.41
Pin Headers	Pines macho/hembra de 2.54mm	1(caja)	8.99	0.69
Caja	Caja negra de ABS de dimensiones: 160x95x55mm	1	11.36	11.36
Soporte	Soporte Cámara Bici	1	17.99	17.99
Soldador eléctrico	Soldador de 26w con punta fina	1	37.05	37.05
Estaño	Estaño para soldadura de 1mm de grosor	1	13.83	13.83
Cableado	Conectores Macho-hembra para realización de puentes	1	8.99	8.99
	Cableado de cobre 0,14mm	3	0.697	2.1
Subtotal				98.32€

Tabla 10 - Coste montaje

4.3 Coste de ingeniería

En este proyecto, tal y como se ha dicho anteriormente, se ha realizado por un ingeniero junior que trabaja para un empresa en una jornada completa de 8h. Teniendo en cuenta, que el sueldo bruto mensual es de 1300€, a este sueldo hay que añadirle el impuesto de la seguridad social que viene desglosado en[30]:

- Contingencias comunes 23,6%
- Pago de las prestaciones por desempleo 5,5%
- Para posibles accidentes de trabajo o enfermedad profesional 3.5%
- Para formación profesional 0.6%
- Para el FOGASA 0.2%

Aplicando todo esto el gasto que le supone a la empresa mensualmente es de **1734.2€.**
Sabiendo que la duración para hacer el proyecto es de (ver tabla 11):

Tarea	Horas
Recerca del material	15
Montaje velocímetro	20
Programación Arduino	160
Programación Processing	240
Comprobación del funcionamiento del proyecto finalizado	20
Total	455

Tabla 11 - Horas necesarias para la realización del proyecto

Como se puede observar para realizar este proyecto se ha necesitado 455h.
Teniendo en cuenta que el ingeniero junior trabaja a jornada completa, es decir, 160h aproximadamente al mes, se ha necesitado 2 meses y 26dias para poder realizar el proyecto. Por lo tanto el coste del el ingeniero junior para la empresa es de **4931.63€.**

4.4 PRESUPUESTO TOTAL

A continuación se muestra el coste total para el desarrollo y realización de este proyecto, teniendo en cuenta que la empresa ha de pagar los costos de la infraestructura (local, agua, electricidad etc.) que se verán representados aplicando un 20% al coste total, además del beneficio industrial que es el porcentaje que el contratista o empresario se marca como beneficio, que suele rondar el 25%.

Coste del velocímetro	54.77
Coste del montaje	98.32
Coste de ingeniería	4931.63
Subtotal	5084.72
costos de la infraestructura	20%
beneficio industrial	25%
TOTAL	7372.84€

Tabla 12 - Coste total

El coste total de este proyecto es **7372.84€,** como se ha podido observar realizar este proyecto no supone un desembolso muy grande, eso es debido al uso de tecnología de bajo coste además del uso de software libre.

CAPÍTULO 5: CONCLUSIONES

5.1 OBJETIVOS CUMPLIDOS

Una vez realizado todo el proyecto puedo afirmar que se ha cumplido los objetivos marcados.
En este proyecto se ha querido demostrar como con el uso de tecnologías de bajo coste se podía realizar un proyecto funcional y muy flexible, debido a que con este mismo proyecto se puede realizar tanto un recorrido en ciudad en bicicleta, como una visita guiada por un museo, una visita por una fábrica etc. sin necesidad de hacer demasiados cambios, tanto en el hardware como en el software.
Además, si valoramos las intenciones personales que tenía al inicio del proyecto, considero que también se han cumplido. Uno de los motivos principales para hacer esta proyecto era la de aprender a usar y profundizar en la programación del software de Arduino y Processing, así como en el montaje de hardware mediante Arduino, y gracias a este proyecto he adquirido unos nuevos conocimientos que pienso que me serán muy útiles de cara al mundo laboral.

5.2 Propuestas de mejora

Una de las mejoras que considero más importantes sería la de ofrecer un recorrido más inmersivo, y esto se podría lograr con el uso de gafas de realidad virtual.
Processing tiene bibliotecas específicas para el uso de la realidad virtual que nos permitiría implementar esta mejora. Los únicos elementos que se tendrían que añadir para poder realizar esto, sería una cámara para poder grabar videos en 360° y un casco de realidad virtual. También sería recomendable cambiar el microcontrolador por otro más potente, con mayor número de I/O y más funciones, debido a que el microcontrolador Arduino UNO R3 se puede quedar corto si se quieren añadir más elementos.

5.3 Valoración personal

Personalmente este proyecto me ha supuesto todo un reto realizarlo, ya que inicialmente no sabía programar en Processing y en Arduino.
Pero con paciencia y esfuerzo y con el soporte de muchas personas (citadas en los agradecimientos) he conseguido realizar el proyecto.
A nivel personal, ha sido un proyecto que he disfrutado realizándolo y me ha resultado muy satisfactorio cuando he visto el resultado final.

Capítulo 6: Bibliografía

6.1 Referencias webs

[1] Características Software Arduino [en línea] (Consulta 01/09/2018)
http://arduino.cl/que-es-arduino/

[2] Explicación Microcontrolador [en línea] (Consulta 01/09/2018)
https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

[3] Qué es Arduino [en línea] (Consulta 01/09/2018)
https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

[4]Características Arduino UNO R3 [en línea] (Consulta 04/09/2018)
https://www.infootec.net/arduino/

[5]Función pines Arduino UNO R3 [en línea] (Consulta 08/09/2018)
https://www.arduino.cc/en/Reference/Board

[6] Esquema pines Arduino [en línea] (Consulta 08/09/2018)
Infootec.net

[7]Librerías Arduino [en línea] (Consulta 15/09/2018)
https://playground.arduino.cc/Main/LibraryList/

[8]Esquema eléctrico potenciómetro [en línea] (Consulta 21/09/2018)
https://www.luisllamas.es/lectura-de-un-potenciometro-con-arduino/

[9]Función del sensor de efecto Hall [en línea] (Consulta 30/09/2018)
https://www.luisllamas.es/detectar-campos-magneticos-con-arduino-y-sensor-hall-a3144/

[10] Datasheet sensor efecto Hall [en línea] (Consulta 30/09/2018)
https://www.elecrow.com/download/A3141-2-3-4-Datasheet.pdf

[11] Imán de neodimio [en línea] (Consulta 01/09/2018)
https://es.wikipedia.org/wiki/Im%C3%A1n_de_neodimio

[12]Función de los pines de la pantalla LCD 1602 [en línea] (Consulta 06/10/2018)
https://lastminuteengineers.com/arduino-1602-character-lcd-tutorial/

[13]Foro Arduino [en línea] (Consulta 12/10/2018)
https://forum.arduino.cc/index.php?board=32.0

[14]Creación velocímetro [en línea] (Consulta 29/09/2018)
http://sebalabs.blogspot.com/2015/10/arduino-12-parte-1-velocimetro-de.html
https://www.instructables.com/id/DIY-bike-tachometer/
http://www.trescarriles.com/cuentarrevoluciones.html

[15] Qué es Processing
https://es.wikipedia.org/wiki/Processing

[16]Librerías Processing [en línea] (Consulta 02/10/2018)
https://processing.org/reference/libraries/

[17] Funciones Processing [en línea] (Consulta 04/10/2018)
https://processing.org/reference/

[18]Foro Processing [en línea] (Consulta 11/10/2018)
https://forum.processing.org/two/

[19]Información Comunicación vía puerto serie [en línea] (Consulta 19/10/2018)
https://www.luisllamas.es/arduino-puerto-serie/

[20] Diagrama del puerto serie [en línea] (Consulta 21/10/2018)
https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/

[21] Envio y lectura de datos via puerto Serie [en línea] (Consulta 22/10/2018)
https://www.arduino.cc/reference/en/language/functions/communication/serial/

[22] Comunicación vía puerto serie [en línea] (Consulta 23/10/2018)
http://panamahitek.com/serial-arduino-introduccion/

[23] Explicación función dtostrf en línea] (Consulta 28/10/2018)
https://programarfacil.com/blog/arduino-blog/conversion-de-numeros-a-cadenas-en-arduino/

[24]Explicación Resistencias Pull-up/Pull-down [en línea] (Consulta 04/11/2018)
https://programarfacil.com/blog/arduino-blog/resistencia-pull-up-y-pull-down/

[25]Función trim Processing [en línea] (Consulta 22/11/2018)
https://processing.org/reference/trim_.html

[26] Librería org.gstreamer.elements.PlayBin2 [en línea] (Consulta 28/11/2018)
https://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/gst-plugins-base-plugins-playbin.html

[27]Github Arduino/Processing [en línea] (Consulta 10/12/2018)
https://github.com/search?q=Processing

[28]Uso de los Timer Arduino [en línea] (Consulta 14/01/2019)
http://rufianenlared.com/timers/

[29]Biblioteca Interfascia de Processing [en línea] (Consulta 09/02/2019)
http://interfascia.berg.industries/

[30]Datasheets componentes electrónicos [en línea] (Consulta 25/03/2019)
https://www.elegoo.com/

[31]Consulta para realización presupuesto [en línea] (Consulta 04/05/2019)
https://www.sdelsol.com/blog/pymes/cuanto-cuesta-un-trabajador/